Coding Practices

Description

This content area describes methods, techniques, processes, tools, and runtime libraries that can prevent or limit exploits against vulnerabilities. Each document describes the development and technology context in which the coding practice is applied, as well as the risk of not following the practice and the type of attacks that could result.

Overview Articles

Naam	Tijdstip aanmaak versie	Abstract
Coding Practices	23/05/06 16:21:21	Most software vulnerabilities are the result of small but reoccurring programming errors that could be easily avoided if programmers learned to recognize them and understand their potential harm. In particular, the C and C++ programming languages have proved highly susceptible to these classes of errors. This knowledge area of the Build Security In web site describes coding practices that can be used to mitigate against these common problems in C and C++.

Most Recently Updated Articles [Ordered by Last Modified Date]

Naam	Tijdstip aanmaak versie	Abstract
strlcpy() and strlcat()	24/05/06 11:02:20	The standard C library includes functions that are designed to prevent buffer overflows, particularly `strncpy()` and `strncat()`. These universally available functions discard data larger than the specified length, regardless of whether it fits into the buffer. These functions are deprecated for new Windows code because they are frequently used incorrectly.
Runtime Protection	24/05/06 11:01:54	There are a number of runtime solutions that can detect stack

		corruption and buffer overruns or guard against attacks. These solutions typically terminate the program when an anomaly is detected, preventing the execution of arbitrary code.
Coding Practices	23/05/06 16:21:21	Most software vulnerabilities are the result of small but reoccurring programming errors that could be easily avoided if programmers learned to recognize them and understand their potential harm. In particular, the C and C++ programming languages have proved highly susceptible to these classes of errors. This knowledge area of the Build Security In web site describes coding practices that can be used to mitigate against these common problems in C and C++.
Consistent Memory Management Conventions	25/04/06 10:23:44	The most effective way to prevent memory problems is to be disciplined in writing memory management code. The development team should adopt a standard approach and apply it consistently.
Guard Pages	4/04/06 14:15:52	Automatic allocation of additional inaccessible memory during memory allocation operations is a technique for mitigating against exploitation of heap buffer overflows. These guard pages are unmapped pages placed between all memory allocations of one page or larger. The guard page causes a segmentation fault upon any access.

All Articles [Ordered by Title]

Naam	Tijdstip aanmaak versie	Abstract
Arbitrary Precision Arithmetic	21/03/06 16:57:25	There are many arbitrary precision arithmetic packages available, primarily for scientific

		computing. However, arbitrary precision arithmetic can solve the problem of integer type range errors resulting from a lack of precision in the representation.
C++ std::string	21/03/06 17:29:28	C++ programmers have the option of using the standard std::string class defined in ISO/IEC 14882. The std::string generally protects against buffer overflow.
Coding Practices	23/05/06 16:21:21	Most software vulnerabilities are the result of small but reoccurring programming errors that could be easily avoided if programmers learned to recognize them and understand their potential harm. In particular, the C and C++ programming languages have proved highly susceptible to these classes of errors. This knowledge area of the Build Security In web site describes coding practices that can be used to mitigate against these common problems in C and C++.
Compiler Checks	21/03/06 17:01:05	In a perfect world, C and C++ compilers would identify the potential for exceptional conditions to occur at runtime and provide a mechanism (such as an exception, trap, or signal handler) for applications to handle these events. Unfortunately, the world we live in is far from perfect. This article provides a brief description of some of the compiler capabilities that exist today.
Consistent Memory Management Conventions	25/04/06 10:23:44	The most effective way to prevent memory problems is to be disciplined in writing memory management code. The development team should adopt a standard approach and apply it consistently.
fgets() and gets_s()	28/03/06 12:17:45	The `gets()` function is a

		common source of buffer overflow vulnerabilities and should never be used. The `fgets()` and `gets_s()` functions each offer a more secure solution.
Guard Pages	4/04/06 14:15:52	Automatic allocation of additional inaccessible memory during memory allocation operations is a technique for mitigating against exploitation of heap buffer overflows. These guard pages are unmapped pages placed between all memory allocations of one page or larger. The guard page causes a segmentation fault upon any access.
Heap Integrity Detection	21/03/06 17:35:33	This article describes a system to protect the glibc heap by making modifications to the chunk structure and management functions.
memcpy_s() and memmove_s()	21/03/06 17:35:57	Substituting the `memcpy_s()` and `memmove_s()` functions for the `memcpy()` and `memmove()` functions can help guard against software vulnerabilities.
Null Pointers	21/03/06 17:29:00	One obvious technique to reduce vulnerabilities in C and C++ programs is to set the pointer to null after the call to `free()` has completed.
OpenBSD	21/03/06 17:37:16	The OpenBSD UNIX variant was designed with an additional emphasis on security. In particular, OpenBSD adopted phkmalloc and adapted it to support guard pages and randomization.
OpenBSD's strlcpy() and strlcat()	28/03/06 12:18:26	Many UNIX variants provides the `strlcpy()` and `strlcat()` functions to copy and concatenate strings in a less error-prone manner.

Phkmalloc	28/03/06 11:52:09	Phkmalloc is an alternative dynamic memory management function that was by written by Poul-Henning Kamp for FreeBSD in 1995-1996 and subsequently adapted by a number of operating systems, including NetBSD, OpenBSD, and several Linux distributions.
Randomization	28/03/06 12:18:48	Randomization works on the principle that it is harder to hit a moving target. Addresses of memory allocated by `malloc()` are fairly predictable. Randomizing the addresses of blocks of memory returned by the memory manager can make it more difficult to exploit a heap-based vulnerability.
Range Checking	28/03/06 11:55:50	Integer range checking, if implemented correctly, can eliminate vulnerabilities resulting from integer overflow, truncation, and sign errors.
Runtime Analysis Tools	28/03/06 11:58:18	Runtime analysis tools that detect memory violations are helpful in eliminating memory-related defects that can lead to heap-based vulnerabilities. To be effective, the tools must be used with a test suite that evaluates failure modes as well as planned user scenarios.
Runtime Protection	24/05/06 11:01:54	There are a number of runtime solutions that can detect stack corruption and buffer overruns or guard against attacks. These solutions typically terminate the program when an anomaly is detected, preventing the execution of arbitrary code.
Safe Integer Operations	28/03/06 12:02:28	Integer operations can result in error conditions and lost data, particularly when inputs to these operations can be manipulated by a malicious user. A solution to

		this problem is to use a safe integer library for all operations on integers where one or more of the inputs could be influenced by an untrusted source.
SafeStr	28/03/06 12:03:42	The C String Library (SafeStr) from Messier and Viega provides a rich string-handling library for C that has secure semantics yet is interoperable with legacy library code in a straightforward manner.
strcpy_s() and strcat_s()	28/03/06 12:19:30	The `strcpy_s()` and `strcat_s()` functions are defined in ISO/IEC TR 24731 as a close replacement for `strcpy()` and `strcat()`. These functions have an additional argument that specifies the maximum size of the destination and also include a return value that indicates whether the operation was successful.
strcpy() and strcat()	28/03/06 12:09:32	The `strcpy()` and `strcat()` functions have been villainized as a major source of buffer overflows, and there are many mitigation strategies that provide more secure variants of these functions. However, not all applications of `strcpy()` are flawed.
strlcpy() and strlcat()	24/05/06 11:02:20	The standard C library includes functions that are designed to prevent buffer overflows, particularly `strncpy()` and `strncat()`. These universally available functions discard data larger than the specified length, regardless of whether it fits into the buffer. These functions are deprecated for new Windows code because they are frequently used incorrectly.
strncpy_s() and strncat_s()	28/03/06 12:09:02	The `strncpy()` and `strncat()` functions are a source of buffer overflow

		vulnerabilities. The `strncpy_s()` and `strncat_s()` functions are defined in ISO/IEC TR 24731 as drop-in replacements for `strncpy()` and `strncat()`.
strncpy() and strncat()	28/03/06 12:11:49	The standard C library includes functions that are designed to prevent buffer overflows, particularly `strncpy()` and `strncat()`. These universally available functions discard data larger than the specified length, regardless of whether it fits into the buffer. These functions are deprecated for new Windows code because they are frequently used incorrectly.
Strong Typing	28/03/06 12:12:42	One way to provide better type checking is to provide better types. Using an unsigned type, for example, can guarantee that a variable does not contain a negative value. However, this solution does not prevent overflow or solve the general case.
Strsafe.h	28/03/06 12:13:39	Microsoft provides a set of safer string handling functions for the C programming language called Strsafe.h. These functions are intended to replace their built-in C/C++ counterparts, as well as any legacy Microsoft-specific string handing functions.
Vstr	28/03/06 12:15:12	Vstr is a string library optimized to work with `readv()`/`writev()` for input/output. For example, you can `readv()` data to the end of the string and `writev()` data from the beginning of the string without allocating or moving memory. This also allows the library to work with data containing multiple zero bytes.
Windows XP SP2	28/03/06 12:16:10	Different versions of the Windows operating systems

		contain different implementations of the heap. The Windows XP SP2 release has two significant improvements over earlier heap implementations that make it more difficult to exploit.

Velden

Naam	Waarde
Categories	knowledge